Optimal Distributed Resource Allocation

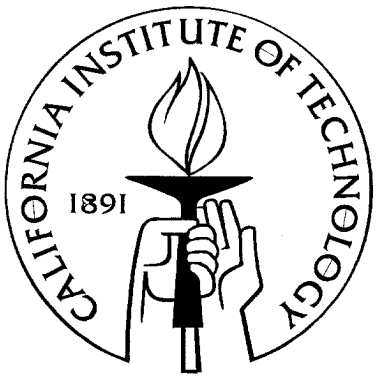
Thesis by

Roman Ginis

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science



California Institute of Technology

Pasadena, California

1999

(Submitted June 11, 1999)

Abstract

We present and explore the problem of automatic distributed resource allocation for a large scale system operating on the basic principles of a free market economy. We model such environment as a resource allocation problem where the producers and consumers are numerous distributed entities controlled by different interests that trade resource time to achieve their goals and maximize their individual objective functions.

A consumer specifies her problem in terms of a graph of resource types that describes temporal relationships between the resources. A resource type is expressed by a predicate that establishes the necessary properties that a resource needs to satisfy to be used in a solution. The consumer also specifies an objective function in terms of the resource type attributes, that needs to be maximized while searching for a solution. A feasible reservation on the part of a consumer, is a set of resources each of which satisfies the required constraints of its type and is available at such times as dictated by the temporal execution order specified by the graph.

We present two polynomial-time algorithms for finding feasible and optimal reservation plans. We also introduce a method to perform a distributed atomic transaction to commence the reservation of the chosen resources. The transaction method is based on the call-option financial instrument that is well suited for the problem.

Finally we show how to use these algorithms to solve the problems in crisis management applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Optimal Resource Allocation is an age-old problem. Human societies have been solving resource allocation problems since the days they first began to organize complex activities that depend on concurrent participation of many individuals. Today, in the age of the global economy, the problem is orders of magnitude more complex as millions of businesses and individuals around the globe are competing, cooperating and simply doing business in an environment that brings them closer than ever. The resources of interest are people, machines, goods, services and information. They are connected by links that communicate literally at the speed of light and their state, the state of the world, changes drastically from second to second. The number of resources at one's disposal is very large and the permutations in which one can choose to use them is enormous. The fundamental problem of optimal resource allocation is: how to choose and reserve the best resources for a job and satisfy the important constraints including cost, time and quality.

An instance of this problem faced by a typical traveler is as following: "A traveler is planning a trip to Paris and needs to reserve some airline seats (to and from), a hotel and a car. She constrains her trip to commence in a 5 day time frame. Furthermore, she defines an objective function that depends on the quality of the airplane seats, the quality of the hotel, the size of the rental car and a relative cost of $2,000. Her problem is to find a reservation plan consisting of the tickets, a hotel and a car that maximizes the objective function, satisfies the constraints, and that can be scheduled."

An important property of this problem is that the resources involved are not owned or controlled by a single interest. As a result, the common goals of existing resource management solutions such as 'maximal job throughput' or 'maximal concurrency' or 'utilization fairness' do not apply to this problem because they are not of primary interest to either the resource producers or the consumers. Instead, each participating entity is interested in maximizing its own optimization function independent of anyone else while trading the resources in a "free-market."

The other important properties of the problem are 1) its scale – literally millions of producers and

consumers, 2) the physically distributed nature of the resources, 3) their diversity and heterogeneity, and 4) the high volatility in their availability and costs. This optimal resource allocation problem is a distributed computing problem (2 and 3); it has the scale of the Internet (1), some characteristics of the stock market used for resource exchange (4) and introduces an additional new requirement of the temporal relationships between resources, e.g., the hotel reservation must begin after the flight arrival.

At this time we do not know of any automation that can solve this class of problems in general. In this thesis we introduce and investigate the free-market resource allocation problem and present polynomial time algorithms for finding feasible and optimal solutions that can be of significant practical use. We demonstrate the utility of these on a resource scheduling example for crisis management applications.

1.2 Resource Allocation Concepts

A *resource* is a source of information or an entity that can perform some function. It can be scheduled or reserved for a duration of time relative to some "universal" clock and it has a unique name – a URL (Universal Resource Locator). Furthermore, it is supervised by a software 'manager' that enables other entities to reserve, release the resource and to determine when the resource is available. An example of a resource is a doctor, whose software manager is a program that manages her schedule.

A *consumer* is any entity that reserves one or more resources to accomplish some goal. A hospital is a consumer when it schedules some doctors to perform an operation, some nurses to care for the patient afterwards, and a driver to take the patient home.

1.2.1 State of the art

Much of the computer science research related to resource management up to now has been focused on the efficient allocation of resources in individual computer systems (such as memory, CPUs and storage) and in distributed networks of computers that agree on some common allocation goals. The bulk of the work has been along two lines – outline the strategies and algorithms that one could implement on a system to achieve some properties; or propose an infrastructure (primarily for distributed computing) that makes many computers implement the same algorithms, communicate and exchange tasks and resources. For example, the Globus project [6] infrastructure makes it possible to run large jobs on many computers and helps with the reservations of all the necessary resources. The HeiRAT [3] quality of service management system allows multiple resources to be scheduled on a distributed system to guarantee levels of quality of service to system tasks. Such algorithms usually try to achieve the following qualities: maximize job throughput, maximize

concurrency, ensure fairness among participants, and improve overall system performance.

These properties are of great interest when the collaborating parties own the resources and would like to share them in a fair manner. For example, if a company owns several mainframes in a distributed network, they would naturally want these machines at peak performance, with fair execution of most jobs (no jobs get left unfinished forever) and with maximum overall job throughput. However, when the participating producers and consumers each owned by different entities and having different and possibly divergent optimization functions, are buying and selling resource time in a free market, the maximization of the traditional qualities above is no longer important or perhaps even desired.

The free market resource model has properties that are uncommon for typical distributed computing problems. For example, it is acceptable for resources to be underutilized. There is no fairness: producers charge prices that make sense from their economic calculations and if a consumer cannot pay, then it cannot reserve and execute. The entities never have "total knowledge" of the state of the system. The system state changes often and the participants find out about the states of others only through message-based interactions that are of interest to them.

Because of the fundamental difference of the properties of the free market resource allocation model from the traditional models, the bulk of existing algorithms for resource management either maximize the 'wrong things' or have their assumptions invalidated. For example, many single-computer algorithms require the total knowledge of the system state, including what processes are running and what resources they are using at any given time. This is not feasible to achieve in a large distributed system without substantial sacrifices in performance.

1.2.2 Why this model and why now?

We believe that the resource allocation model presented here would work well in a connected free market economy such as the economy we currently have in the US. It would allow a more automated exchange of services and offer more options and guarantees to the participants. Implementations of this model could yield new applications in any branch of industry that deals with dynamic allocation of resources. For instance, it could be used in electronic commerce and crisis management applications.

This model can be implemented with today's technologies. Currently every major corporation is connected to the Internet. Thousands of them offer their services and items for sale on-line. Besides the Web, several distributed communication infrastructures are widely available, including Object Management Group's CORBA [8], Sun Microsystems Java JINI [7], DCOM+ from Microsoft [4], Caltech Infospheres [1] and many others. These infrastructures facilitate the communication between distributed computer systems using messages and offer services such as Transactions, Security, Persistence and Directories to applications that use them. They work across platforms and take care

of all incompatibilities between the systems. Some are already deployed in the field and are quite mature. The resource allocation algorithms described in this thesis can be implemented on top of these technologies.

In the near future almost every business will likely need not only an Internet 'presence' but will have to offer its services for sale on the Internet as well. The number of on-line resources for rent or sale is currently growing exponentially and shopping for them will become ever harder as more choices become available. An automated approach for resource reservations like the one proposed in this thesis will become a necessity to transact with any significant business, perform a computation or gather information on-line.

1.3 A simple example

We now present a more precise specification of the "Trip to Paris" problem described earlier and show the flavor of the solutions that the algorithms in this thesis can offer.

A consumer is planning a trip during which certain resources have to be used in a particular order. We represent this order by a directed acyclic graph G (e.g., Figure 1.1) where the nodes represent the resource types that need to be reserved and the edges specify the order in which these resources will be used.

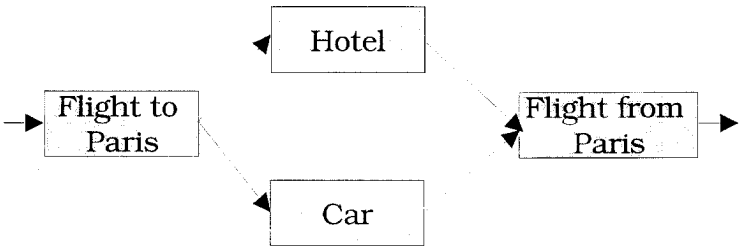


Figure 1.1: "Trip to Paris" Computation Graph

The additional constraints include the maximum trip duration of 5 days and the earliest start time = $now = 0$. There is an objective function given as:

$$F = 0.5 \times flight.quality \times 2 + 0.8 \times hotel.stars + 0.2 \times car.size + 1.3 \times (\frac{\$2,000}{price})$$

where the $0.5, 0.8, 0.2$ and 1.3 are the parameters that reflect the priorities between the resource

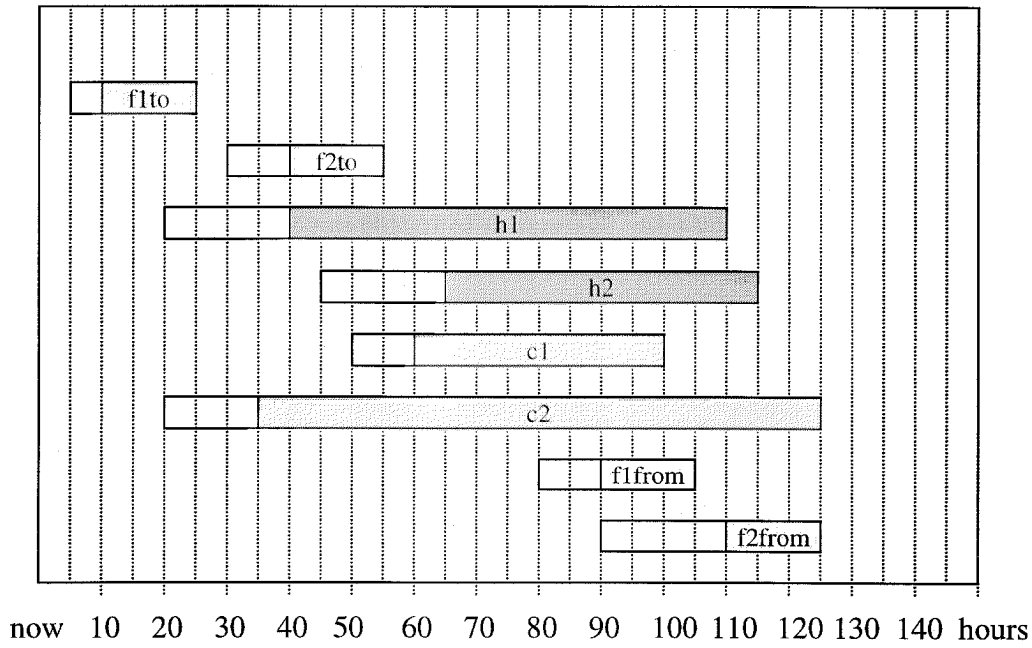


Figure 1.2: "Trip to Paris" Resources

attributes. Let the resource utilization horizon be

$$H = \text{maximum duration - earliest start time} = 5 - 0 = 5 \text{ (days)}$$

The reservation problem is solved in four stages. The consumer will 1) communicate with a directory service and the resources to discover the possible resource options, 2) perform a feasibility analysis to find out whether at least one feasible schedule exists, 3) execute a planning algorithm to construct an optimal reservation plan, and finally 4) make the reservation of the optimal plan as an atomic transaction.

For simplicity we will assume here that stage 1 has already commenced and that all the resource options have been acquired. Assume that Figure 1.2 describes these resource bids. The labeled blocks in the figure represent the various resource options that the client has discovered. The clear part of each block reflects the flexibility of the resource's reservation start time. The shaded part is the time that the resource needs to perform the required task. If a resource is picked for a reservation, it must be reserved for the duration of the shaded region, and start any time in the white region.

For example, the resource **f1to** in the figure represents one possible flight to Paris. It can start any time between 5 and 10 hours from now and will require 15 hours for the journey.

The chart below (Table 1.1) lists the additional information about each of the resources in the

figure that is important in computation of the objective function F. For instance, the flight **f1to** happens to cost $400 and have a quality rating of 2. Since both the cost and the quality of the flight are factors in the given objective function, a value could be computed and assigned to this flight if it were picked for a reservation.

	Flight to Paris	**Hotel**	**Car**	**Flight from Paris**
1	f1to: $400 quality 2	h1: $800 stars 3	c1: $100 size 3	f1from: $300 quality 1
2	f2to: $300 quality 1	h2: $600 stars 2	c2: $200 size 4	f2from: $500 quality 2

Table 1.1: Resource attributes

From these schedules one can construct the following *feasible* reservation plans:

Plan	**Value**
f1to, h1, c1, f1from	6.13
f1to, h1, c1, f2from	5.94
f1to, h2, c1, f2from	5.83
f1to, h2, c2, f2from	5.93
f1to, h1, c2, f2from	6.73
f2to, h2, c1, f2from	5.43

Table 1.2: Feasible plans

The algorithms we will present would first check if a feasible plan exists and then produce the optimal plan (f1to, h1, c2, f2from) with the value of 6.73. If $|G|$ is the number of nodes in the computation graph, $|R|$ is the total number of resources and H the time frame horizon then the algorithm's worst case computation time would be $O(|R| * |G| * H^2)$.

The final stage 4 would commence the reservation of the optimal plan as an atomic transaction.

1.4 Overview

The remainder of the thesis is structured as follows. Chapter 2 contains the background and the basic concepts of distributed computation that underlines our discussion. In Chapter 3 we introduce our solution model that describes the involved entities and their interaction. We propose and prove the solutions formally in Chapters 4 and 5. Finally, Chapters 6 and 7 contain a discussion about sample applications and future work.

Chapter 2

Model of Computation

We represent the resource allocation problem in a formal model so that we can reason about it precisely. The model defines the properties of its constituent entities and their interaction. Our resource allocation problem deals with entities (producers and consumers) that are physically and possibly geographically distributed. A model that has been developed to apply to this case is the distributed system model. A distributed system is a set of computers that do not have shared memory, yet appear to users as a single computer.

2.1 Properties of the Distributed System Model

A distributed system consists of physically independent entities connected by a communication infrastructure (network). Its primary properties are as follows:

1. **Concurrent execution of all entities**

   Every entity in the system is a separate computation that executes concurrently with the other entities.

2. **Communication infrastructure between entities**

   The entities can communicate with each other using messages. There are two primary communication paradigms: synchronous and asynchronous. In synchronous communication, the entity sending a message stops its computation and waits for a reply from the receiving entity. In the asynchronous communication paradigm each entity can be thought of as having two mailboxes: an inbox and an outbox where the messages are queued until they are processed by the receiving entity's logic (in the case of the inbox) or delivered to their destination (in the case of the outbox). It has been shown that the asynchronous communication can be implemented on top the TCP/IP networks [1].

3. **Old knowledge**

   An important property of a distributed system is that each entity knows only its own state.

In particular, a given entity does not have the current state of other entities. This is a direct result of the communication delay between entities, their separate computations and clocks discrepancy. What an entity *could* know is some earlier state of another entity (e.g., from an earlier communication). Thus, all distributed computation is done using "old" knowledge about the states of the other entities in the system. (This will become important when we discuss the distributed atomic transactions in Chapter 4).

2.2 Dealing with Real-Time constraints

All transactions between entities in our model have temporal constraints and refer to future events. For example, a consumer can ask for the hotels available "during the next 2 weeks" or to make a reservation "within 24 hours." For this reason, all participating entities must have access to a "universal" clock so that the temporal references would mean the same to everyone.

Earlier we have stated that entities in distributed systems do not share knowledge. A "universal" clock is knowledge shared by everyone. Therefore, though we cannot achieve a "true" universal clock, an approximate one is sufficient for our problem. There are hardware and software solutions in existence that can guarantee very tight clock synchronization with high degree of probability [5].

2.3 Instances of distributed systems

Currently, the most ubiquitous distributed system is the 'Internet.' It consists of a myriad of computers connected by a network and communicating with a specific network protocol: TCP/IP. There are numerous software packages that utilize the network and the protocol to communicate data in certain standardized formats such as *HTML*, *XML* and higher level protocols such as *HTTP* and *FTP*.

There are more advanced distributed systems that offer additional services and capabilities to many applications. One such is Object Management Group's CORBA (Common Object Request Broker Architecture) [8] that abstracts the network to client/server interactions between entities modeled using the Object Oriented design principles. Another, is Caltech Infospheres Infrastructure [1] that uses peer-to-peer communications between entities, and offers additional capabilities for designers to better compose, model and reason about distributed object systems.

Chapter 3

A Solution Model

There are three types of entities in our model: producers, consumers and directories. The producer and consumer instances of these entities represent real-world individuals, organizations, machines, services and information. The directories are repositories of information about the producers that enable the consumers and other directories to locate the resources they seek. The "system" for this model is a distributed system of numerous instances of the three types of entities connected by a network.

In this chapter we specify the basic features of each of the entities and restate the problem more formally. The two chapters that follow describe the solution we propose.

3.1 Resources

A *resource* (or producer) is a source of information or an entity that can perform some function. With respect to the system it is defined by the following:

1. A unique Universal Resource Locator (URL) – a name (e.g., a string) that makes it possible for the communication infrastructure to deliver messages to and from this resource.
2. A set of attributes that describe the properties of the resource (e.g., 'high resolution temperature sensor with $\epsilon = 0.01^o$ ')
3. A set of 'available periods' in a form of tuples: (absolute start time, absolute end time)
4. A cost function that gives the price of a reservation depending on the time and the duration.

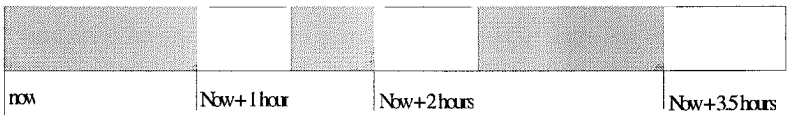
A URL makes a resource 'unique' in the system. Messages sent to a URL arrive to the single resource with which this locator is associated. Likewise, messages received from a particular URL could only have come from a unique source.

The set of properties completely describes the capabilities a resource chooses to expose to the outside world. For example, the properties of a certain fire emergency vehicle could be described as follows:

```
<vehicle><emergency><fire>
    <name> 1978 American Century Series Pumper </name>
    <pump> 1,500 GPM </pump>
    <water tank> 500 Gallon </water tank>
    <water gun> Deck </water gun>
    <generator> 6KW Diesel <generator>
    <flood lights> 500 Watt </flood lights>
    <price> 400 dollars per hour </price>
</fire></emergency></vehicle>
```

The strings in brackets denote attribute names and the strings they enclose are their corresponding values.

The available periods for an emergency vehicle could be visualized as a segment of a time line:



Where the dark regions indicate busy times and clear regions indicate the available times. If a consumer were to ask this resource for availability information, the meaning of 'Now' in the figure above would be the same for the consumer and the resource because of the universal clock service in the environment.

When a resource is 'busy' in a block of time, then only one (the reserving computation) can be using it during that period. The physical resources that can handle many concurrent reservations would represent themselves as many separate instances of the same resource. For example, a hotel would not represent itself as one resource which is either busy or available. Rather, a hotel would be represented by a set of 'room' resources with individual attributes and availability.

3.2 Directories

Directories are databases containing information about resources and their attributes. When a consumer wants to find a resource, it queries a directory in the following form: "select all resources where attributes are Γ." The directory then returns a (possibly empty) set of resource URLs that satisfy the constraints Γ. A consumer can then use these resource locators to negotiate with and reserve the corresponding resources. For instance, a consumer from the "Trip to Paris" example

in the Introduction could make a directory request such as: "select all resources where (hotel is in Paris AND hotel > 3 stars AND hotel.price < $50 per night AND hotel.available within 2 weeks)"

The 'All resources' in the above query refers to all the resources tracked by a particular Directory. If the Directory instances form a network, or a Directory Service, then the query could be propagated to more than one database. Since it is entirely the responsibility of individual resources to register with the directories, no client is guaranteed to find all the resources satisfying its query, even if they do physically exist.

In a practical system, the directory can have additional services such as security and fault-tolerance. The former can control the resource 'visibility' to different consumers, for instance returning the information about some resources but not the others. The later can provide guarantees, such as if a resource satisfies a query, its locator will necessarily be returned, barring the security restrictions.

3.3 Computations

A consumer is a software entity that initiates a *computation.* A computation is a solution to a problem that can be described in terms of a computation graph G (see Figure 3.1) that specifies a temporal relationship between the resources, a set of constraints N on the resource types, an objective function F and a time horizon H during which the problem must be solved.

The nodes of the graph are resource types; the edges represent both the flow of data between the resources (if any) and the execution order. Namely, in the figure, the resource of type A must be used first, followed by B, C and D where the BD sequence can be executing concurrently with C. The resource E will execute only after A-D have completed their computations. Similarly, the data (if any) must flow from A to E along the edges.

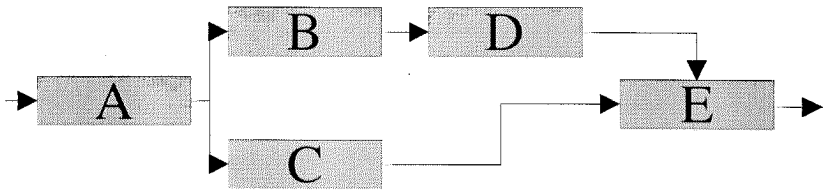


Figure 3.1: A simple computation graph

The constraints placed on the computation consist of the constrains on each resource, for example 'the airplane seats must be at least Business Class' as well as on the entire solution, i.e., 'the trip

must end within 5 days.'

A reservation is a mutual commitment between a specific computation and a specific instance of a resource. This commitment includes the reservation time periods and the information where to retrieve the input data and where to deposit the outputs (if applicable).

The objective function is a representation of the importance of individual resource types in relation to each other. The goal of every computation is to maximize the objective function while satisfying all of its constraints.

The form of the optimization function will likely vary with applications. A taxonomy of useful forms is needed and is a subject of future research.

3.4 Problem statement and Solution Outline

We are now ready to state the problem formally:

3.4.1 Problem

Given a computation defined by the tuple $\{G, N, F, H\}$:

1. reserve a set of resources as an atomic transaction that satisfies G, N on time period H and maximizes F.

2. guarantee that a feasible solution will be found, if one exists.

3.4.2 Solution Outline

We propose a solution that works in four stages. A consumer seeking a reservation would:

1. Communicate with a directory service and the resources to discover the possible resource options

2. Compute to find out if a feasible schedule exists

3. Construct an optimal reservation plan

4. Reserve the optimal plan as an atomic transaction

The stages 1 and 4 deal with all the distributed computing parts of the problem and are described in detail in the next chapter. The stages 2 an 3 are purely computational, containing algorithms that deal with the feasibility and optimality of a fixed set of resources discovered in stage 1. These are discussed in Chapter 5.

Chapter 4

Solution Part 1 - Distributed Interaction and Reservation Transaction

In this chapter we describe the first part of a solution to the problem that we stated in the previous chapter. First, we outline our solution informally, identifying its basic parts. Then we specify each of the entities precisely, using a high level pseudo code that enables us to describe their important behavior and avoid the syntactic details of particular programming languages.

The solution we present consists of the algorithms for the consumer, producer and the directory entities. In this thesis our focus is almost entirely on the consumer. The algorithms for the producer and the directory are little more than skeleton behaviors that a consumer needs to be able to interact with them. In general, algorithms for the producers and the directories could be quite sophisticated. For example, designing the distributed databases for the directories to efficiently store the data about producers is a research field in its own right. One could also invent the producer algorithms to perform elaborate computations to decide what prices to set on their resources. However, in this thesis we choose to concentrate on the consumer resource plan construction and producer-consumer transaction interaction, because we believe these to be the basic parts that are necessary for our model to function.

4.1 Entities (operational overview)

4.1.1 Consumer

A consumer interacts both with the producers and a directory to instantiate a computation. Its behavior consists of the following basic steps:

1. Ask the directory service for information about resources that satisfy some constraints

2. Request availability information from each prospective resource

3. Run an optimization planner and construct feasible plans

4. Make a reservation of the resources in the best plan as an atomic transaction

In the first step the consumer seeks to locate all the resources that it could potentially use in its computation. To do this, it sends the specification of the resources it requires to a Directory entity. The contacted directory returns back to the consumer a set of resource locators (URLs) that satisfy the outlined constraints. If this set is empty (that is, the directory did not find any resources matching the query) then we assume that the client terminates (fails).

In step 2, the consumer directly contacts the resources whose locators it acquired in step 1, asking for their availability information. The resources respond by sending back to the consumer lists of their available time slots. At this stage, the consumer has all the information it needs about each resource to be able to determine which ones it could use best. In step 3, the consumer uses this information to construct possible reservation scenarios (plans) optimized with respect to its objective function. Finally, in step 4, the consumer entity negotiates with the resources chosen in step 3 to perform reservations according to the plans it has constructed.

4.1.2 Directory

A directory entity has a message loop and responds to messages from consumers. The single type of message expected by the directory is a query for resources based on a set of constraints. This is similar to a database lookup, where the consumer specifies the constraints in arbitrary detail and expects the directory to identify all the resources matching them. When a directory receives such a request, it queries its own internal database and possibly the databases of other directories and returns a (possibly empty) set of resources as a result.

4.1.3 Producer

A producer in our model registers with a directory so that consumers could find and consider it for their computations. It then proceeds to wait for a message from a consumer. There are two categories of messages: information request and reservation/release messages. A producer receives the first type when a consumer in its step 2 asks for the resources availability. It replies to that by sending its availability time list. The second message type is invoked when a resource enters a negotiation for a reservation. One type of a reservation message requests a right for a consumer to reserve this resource some time in the future and to hold this right for some small amount of time. The second is the actual reservation request (and a matching reservation release). The two

reservation types are crucial in making a distributed atomic transaction between a consumer and the resource take place.

The last two steps of the consumer, dealing with an optimal resource selection and reservation negotiation will be the topic of our discussion for the remainder of this document. We will discuss consumer-producer reservation transaction (step 4) presently. The optimal resource selection (step 3) is treated separately in chapter 5.

4.2 Distributed Transaction

We now present an algorithm for commencing an atomic distributed reservation transaction between a resource and a consumer. We assume that the consumer is at its final stage, where it has prepared a set of reservation plans (each consisting of multiple resources) that were found to be feasible, and ranked by their objective function values. Its goal is to commence a reservation of at least one of these plans, preferably the one with the highest total objective function value, namely, as close the optimal as possible.

By a plan reservation we mean an atomic operation, such that either all the resources in the plan become reserved or none at all. For instance, if a consumer has a plan for the "Trip to Paris" example, and the return flight were not available when the consumer becomes ready to make a reservation then the consumer would not be interested in a partial reservation of just the unidirectional flight, a hotel and car. Only the reservation of the whole plan is of interest.

The reason why such a reservation commitment is tricky is because the producer and the consumer entities are in a distributed system and the information they have about each other is always aged. In particular, the availability of a resource a consumer schedules in one of its plans might have changed since it was requested. Thus, when the consumer attempts to make a reservation, it would fail for that resource. Since a plan consists of a number of resources that all have to be reserved (or not reserved) together, it is not in consumer's interest to start reserving a plan and failing some time during the process. To remedy this, one needs a mechanism that would give the consumer an insurance that its attempt at reserving a plan would succeed with high probability (it is not possible to offer an absolute guarantee, since any entity in the system could, for example, loose its network connection, rendering it unreachable).

We propose a solution for a distributed atomic transaction based on a financial instrument called **American Call Option** [2]. We define an option in our model as *a right, that can be purchased by a consumer, to reserve a resource for some specific time in the future.* An option has a fixed expiration date T (the date when the right expires), a strike price K (the cost of the actual reservation) and a premium U (the amount a consumer must pay to purchase a right to reserve). A option, when purchased, is binding to the resource for which it was purchased, but it is not binding

to the consumer. By buying an option for a resource, the consumer has a right, not an obligation to reserve that resource.

Using this reservation tool, it now becomes possible for a consumer to purchase short-lived tentative reservation commitments from resources by paying them to hold the time blocks that it is interested in. Now, the algorithm for the atomic reservation transaction can be outlined as follows.

Suppose a consumer C wants to make a reservation of resources in some plan P. Then it performs the following steps:

1. send a message to each resource in P asking to purchase an option for a specific time block
2. wait for a confirmation from all the resources or stop the reservation if a negative response is received or the wait times out with some of the resource confirmations missing.
3. if the reservation stopped in step 2 then go to next plan, otherwise send reservation messages to each resource in P.
4. wait for a confirmation from all the resources or roll-back (cancel all) the reservation if no response is received from any of the resources in the allotted time.

Note that no explicit releasing of resources is necessary, since all reservations are done for a fixed amount of time.

This algorithm ensures an atomic transaction between a consumer and a resource if 1) the conditions for commencing the actual resource reservations in step 3 have been met, 2) the option expiration date for the resources was sufficiently far into to the future for the reservation requests to reach the resources, and finally 3) the infrastructure was able to deliver the messages successfully.

In the sections that follow we give a pseudo-code programs for the entities. The code is a high level implementation of the functionalities that we have discussed.

4.3 Program Notation

As stated in chapter 2, we assume the presence of an infrastructure capable of delivering messages from one entity to the other. Our entities are entirely message-driven: each entity has an event loop constantly waiting for messages. When one arrives, an entity executes its program code to handle the message and immediately returns back to waiting. The messages received while an entity is handling an earlier message are queued in FIFO order and become available to the entity when it returns to the message loop.

We use **receive MessageName(M) time-out in X** expression to denote the readiness of an entity to receive a message of type *MessageName*. Each message can be though of as a structure capable of carrying arbitrary information. When a message is processed (or accepted), its payload

can be found in variable M. The message variable M also contains a special attribute, *sender*, referred to as *M.sender*, that identifies the source of the message. We also assume that a message can carry arbitrarily complex data – a list, a matrix, etc. If a **receive** statement has the **time-out** extension it signifies that the entity executing it should processes messages of type MessageName until X units of time have expired. After that it should stop processing messages of this type and continue its computation. On the other hand, the **receive** statements without the time-out check the message queue of the calling entity and continue immediately if no messages matching their type are present.

We use **send M to E** expression to signify the entity's intent on sending a message M to some other entity E. The **send** command is asynchronous and returns immediately, allowing the sending entity to continue its computation. The **send M to E in X** is an instruction to deliver message M to entity E in X time units. An entity can use this to send alarm messages to itself.

4.4 Programs

The following are the programs for the three entities in our model:

4.4.1 Directory

```
1. wait for messages:
2.     receive ResourceLocationsRequest(M):
3.         R := resources satisfying M.constraints in local database
4.         send ResouceLocators(R.URL) to M.sender
```

4.4.2 Producer

```
1. Let A be a list of available time blocks
2. wait for messages:
3.     receive AvailabilityRequestMessage(M):
4.         send AvailabilityInformation(A) to M.sender;
5.     receive ReserveOptionRequest(M):
6.         if M.time block in A then // if block is available
7.             remove M.time block from A; mark M.time block as optioned;
8.             send OptionExpired(M.time block) to self in M.dt; // set option expiration alarm
9.             charge M.sender for the option
10.            send ReserveOptionCommit(M.time block) to M.sender;
11.        end if
12.    receive OptionExpired(M):
13.        // check if the consumer has reserved its block
14.        if(M.time block is not marked reserved) then
```

```
15.                 mark M.time block as free // option has expired
16.                 add M.time block to A // add it back to the free list
17.                 send OptionExpiredMessage(M) to M.sender
18.             end if
19.         receive ReservationRequestMessage(M):
20.             if (M.time block in A OR M.time block marked optioned) then
21.                 mark M.time block reserved
22.                 charge M.sender for the reservation
23.                 send ReservationSuccessfulMessage(M) to M.sender;
24.             else
25.                 send ReservationFailedMessage(M) to M.sender;
26.         receive ReservationCancellationMessage(M):
27.             if (M.time block marked as reserved) then
28.                 add M.time block to A
29.                 unmark M.time block
30.                 charge M.sender for the cancellation
31.             end if
```

4.4.3 Consumer

```
1. Let Resources be a list.
2. send ResourceLocationsRequest(graph, resource constraints) to Directory
3. wait for messages:
4.     receive ResourceLocators(M) timeout in X:
5.         for R in M.resources:
6.             send AvailabilityRequestMessage to R;
7.         end for
8.     receive AvailabilityInformation (M) timeout in Y:
9.         add M.information to Resources;
10. Plans = Construct Optimal Plans from Resources; // see chapter 5
11. // atomic reservation
12. for plan in Plans do
13.     for resource in plan do:
14.         send ReserveOptionRequest(time block, dt) to resource;
15.     end for
16.     wait for messages:
17.         receive ReserveOptionCommit(M) timeout in X:
18.             mark resource as optioned
19.     if all resources are optioned then:
```

```
20.         for resource in plan do:
21.             send ReservationRequestMessage(time block) to resource;
22.         end for
23.     else
24.         break; // move on to the next plan – this one failed
25.     end if
26.     wait for messages:
27.         receive ReservationSuccessfulMessage(M) timeout in X:
28.             mark resource as reserved
29.     if all resources are reserved then:
30.         SUCCESS
31.         exit; // stop considering any more plans.
32.     else
33.         FAILURE
34.         // cancel all reservations and pay penalties
35.         for resource in plan do:
36.             send ReservationCancellationMessage(time block) to resource;
37.         end for
38.         // move on to the next plan
39.     end if
```

Chapter 5

Solution Part 2 - Reservation Planning and Optimization

In this chapter we deal with the resource selection part of the problem. We assume that the resource bids have already been received and stored in a consumer's memory. Now we can treat the resource selection purely as an optimization problem. We will ignore all the distributed computing aspects and consider the received resource information as the consumer's only knowledge about the world.

We are interested in two problems with respect to resource selection: 1) finding a feasible solution and 2) an optimal solution. We will show an algorithm for finding a feasible solution in polynomial time in the size of the number of resources, that is guaranteed to find a solution if one exists. We also present a polynomial time algorithm for the optimal solution over a discrete time line for the class of computation graphs that we call regular.

5.1 Terminology

Let G be a directed acyclic computation graph. We enumerate the nodes in G and refer to the i^{th} node as G_i (see Figure 5.1).

Definition 1 *Let R be the set of all resource bids.*

1. $Q \triangleq |G|$ *is the cardinality of* G.
2. $NG \triangleq \{i : 1 \leq i \leq Q\}$ *the set of node numbers*
3. $NS \triangleq \{i : G_i \in S\}$ *where* $S \subseteq G$
4. $i \rightsquigarrow j$ *denotes the statement "G_i has an edge to G_j"*
5. $ANCESTORS : NG \longrightarrow 2^{NG}$
6. $ANCESTORS_i \triangleq \{j : j \in NG : j \rightsquigarrow i\}$ *the set of node numbers of ancestors of* i
7. $W = \langle \max i : i \in NG : |ANCESTORS_i| \rangle$, *the 'width' of* G, *which we will use to reason about algorithm complexity.*
8. $R_i \subseteq R \triangleq$ *set of all resource bids from resources that satisfy the constraints of node* G_i.

9. $\mathbf{T} \triangleq \mathbf{R}$ *(reals)*
10. $R^t \subseteq R \triangleq$ *set of resources capable of ending at time* $t \in \mathbf{T}$.
11. $M = \langle \max i : i \in NG : |R_i| \rangle$, *which we will use to reason about algorithm complexity*

We assume the numbers assigned to the nodes satisfy the following predicate:

$$\langle \forall i, j :: i \rightsquigarrow j \implies j > i \rangle \tag{5.1}$$

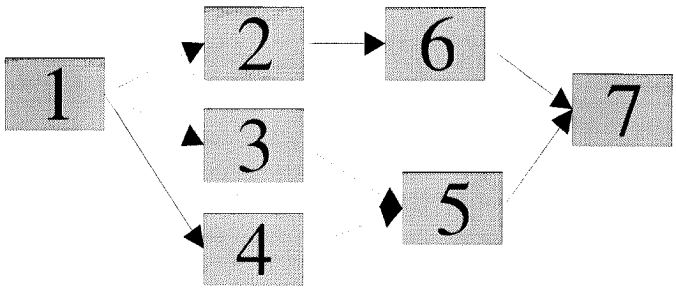


Figure 5.1: A computation graph with Q=7 and W=2

For convenience, from now on we will refer to any set of nodes from G by their corresponding numbers assigned above. For example, the set $\{1, 2, 3\}$ shall represent the set of nodes so enumerated.

Definition 2 *Let* $r \in R$ *and let the following be legal functions on* r*:*

1. $r^{EST} : R \longrightarrow \mathbf{T}$ *returns the earliest start time for* r
2. $r^{LST} : R \longrightarrow \mathbf{T}$ *returns the latest start time for* r
3. $r^{S} : R \longrightarrow \mathbf{T}$ *returns the start time for* r *when* r *is in a plan, otherwise returns* $-\infty$
4. $r^{VAL} : R \longrightarrow \mathbf{R}$ *returns the objective function value of* r
5. $r^{NODE} : R \longrightarrow NG$ *returns the node in NG for which* r *is a resource bid*
6. $r^{D} : R \longrightarrow \mathbf{N}^{+}$ *returns the duration of* r
7. $\perp$: *a special resource such that* $\perp^{EST} = -\infty, \perp^{D} = +\infty$
8. $\Omega : R \times T \longrightarrow R \triangleq \{r, t\} \longrightarrow r' : r = r'$ *except* $r'^{EST} = t$

Definition 3 *(Plan) Let* $S \subseteq G, P_{NS} \subseteq R$. *Then* P_{NS} *is a partial plan for* R *over subgraph* S *if and only if*

$$\langle \forall i \in NS :: \langle \exists r \in P_{NS} : r \in R_i \wedge r^{EST} \leq r^{S} \leq r^{LST} \rangle \rangle \wedge |NS| = |P_{NS}| \tag{5.2}$$

in the special case when $S = G$, *we will omit the subscript NS in* P_{NS} *and call* P *a* ***complete*** *plan.*

Whereas each input resource r has a flexible start time in the range between r^{EST} and r^{LST}, assigning a resource to a plan fixes its start time r^S such that $r^{EST} \leq r^S \leq r^{LST}$.

Definition 4 *Let P_{NS} be a partial plan. We define the following functions:*

1. Π *is the set of all partial plans on* R
2. $P_{i..j} : NS \times NS \longrightarrow 2^R$ *is the set of resources from a complete plan* P *that stands for nodes* i *through* j.
3. $P_{NS \cdot i} : NS \times \Pi \longrightarrow R$ *refers to the resource for node* G_i *in the partial plan* P_{NS}
4. $P^E_{NS \cdot i} : NS \times \Pi \longrightarrow \mathbf{T} \triangleq P^S_{NS \cdot i} + P^D_{NS \cdot i}$ *returns the end time of the resource for node* G_i *in the partial plan* P_S
5. $P^S_{NS} : NS \longrightarrow \mathbf{T} \triangleq \langle \min\, i : i \in NS : P^S_{NS \cdot i} \rangle$
6. $P^E_{NS} : NS \longrightarrow \mathbf{T} \triangleq \langle \max\, i : i \in NS : P^E_{NS \cdot i} \rangle$
7. *for the case when* $S = G$ *the above functions are called:* P_i, P^S_i, P^E_i, P^S *and* P^E *respectively.*

Axiom 1 *Let $r \in R$ and let $P_{NS} \in \Pi$ then:*

1. $r^{EST} \leq r^{LST}$
2. $\langle \forall i : \perp \in R_i \rangle$

Definition 5 *The objective function value of a partial plan is the sum of the values of the plan's resources:*

$$P^{VAL}_{NS} = \sum_{k \in NS} P^{VAL}_{NS \cdot k} \tag{5.3}$$

5.2 Solutions

5.2.1 Feasibility

When a consumer receives a set of resource bids, it is not immediately obvious whether any of them could be composed into a feasible solution. The reason is that any resources that form a solution also form a schedule that specifies precisely when each of them must start. Because each resource has a flexible start time which could be any subset of $(0, H)$, it is not sufficient to just have one of each of the required resources. The consumer's computation graph dictates exactly when the resources must start and end with respect to each other. A feasible reservation is such a chosen and scheduled set of resource whose start and end times match up in way that satisfies the graph.

For a given set of resource bids R we would like to know if it is possible to find a feasible plan, i.e., a subset of R consisting of exactly one resource for each node in the computation graph G and having such start and end times as to satisfy the temporal order of execution imposed by G.

Definition 6 *(Feasibility):*

A partial plan P_{NS} over the set of nodes $NS \subseteq NG$ is called feasible if and only if

$$\langle \forall\, i,j : i,j \in NS : i \rightsquigarrow j \implies P^{E}_{NS \bullet i} \leq P^{S}_{NS \bullet j} \rangle \tag{5.4}$$

(Recall that $P_{NS \bullet k}$ stands for the resource in a partial plan P_{NS} corresponding to node number k in the computation graph G. Also, the superscripts S and E indicate the start and the end time of this resource respectively).

Now we can state the feasibility problem more precisely:

Problem 1 *Given a computation graph G and a set of resource bids R find a complete feasible reservation plan if one exists.*

We are interested in constructing an algorithm that will find a feasible plan if one exists and guarantee that one does not exist if it fails.

5.2.1.1 The Solution: Informal Description

The search space of all possible reservation plans is very large and no brute force solution of examining each one could compute in a reasonable time. However, here we are interested in finding at least one feasible plan. The basic idea of our algorithm is to construct the plan resource by resource, somehow selecting one for each node in the graph starting with the smallest node number. For example, for the graph in Figure 5.1 we would first find a resource for node 1, then 2, etc.

The nodes in the graph were enumerated in such a way that if there is an edge between two nodes, then the source node has the smaller number of the two. This allows us to reason that when we are at the stage in plan construction when we are looking for a resource for node i then we have already found and scheduled all the ancestors of node i. Namely, we have a partially completed set of resources (or a partial plan) up to node $i-1$ in which we know the start and end times of each resource, and in particular, of all the nodes that happen to be parents of i. For instance, if we are constructing a plan for the graph in Figure 5.1, and looking for a resource for node 5, then we can assume that we have a partial plan that includes the resources for nodes 1..4.

Suppose we have many resources for node i. The key idea of the algorithm is to choose the resource for i that 1) starts after all of its parents have completed, which is the requirement of the graph, (for node 5 in the example, the resource for 5 would have to start after the resources for 3 & 4 have completed) and 2) ends at the earliest possible time with respect to all other such resources for i. The reason for choosing the resource that ends the earliest, is that it would allow **maximum flexibility** in choosing the subsequent resources for the node numbers greater than i. Thus, if at each stage in the feasible plan construction, we choose the resource that allows maximum

possibilities for the next stage, then we can be certain that we have not missed a possibility when we are finished with the process. In the example, choosing a resource for 5 that ends the earliest of all resources for 5 allows the most number of resources for 7 to be considered.

The remainder of the construction and the proof of correctness are by induction on i.

5.2.1.2 The Solution Strategy: Formal Description

The algorithm can be described by the following function:

$$REET\_FILTER : 2^R \longrightarrow (2^{R'}, 2^{R'})$$

where R is the initial set of resources, R' is a set of transformed resources and $REET$, which is the name of the second output of $REET\_FILTER$, is a set of resources constituting a feasible plan. Though Problem 1 could be satisfied by just one feasible plan, we designed our solution function to output a feasible plan, $REET$, and an additional set R'. It may be possible to construct many feasible plans from R. The set R' contains the resources that could be part of at least one feasible plan.

The function has the word 'filter' in its name because it can be thought of as filtering the resources in R into R' leaving out all those that could not be a part of any feasible plan due to incompatible start and/or end times. Once the set R' is constructed it forms the basis for one special plan called $REET$ that stands for *Resources with the Earliest End Times.*

For convenience the set R' inherits all the functions legal on R (e.g., R'_i stands for the set of resources in R' for node G_i). Also, in the following formulae, since $REET$ is a complete plan, $REET_i$ refers to the resource in $REET$ for node G_i.

Definition 7 *We define $REET\_FILTER$ to have the following postcondition:*

$$\begin{aligned}
RF0 &: \langle REET \text{ is a complete plan for } R' \rangle \\
RF1 &: \langle \forall\, i \in NG :: \langle \forall\, r' \in R'_i :: \langle \exists r : r \in R_i : r' = \Omega(r, \max(r^{EST}, MAR_i)) \wedge MAR_i \leq r^{LST} \rangle \rangle \rangle \\
RF2 &: \left\langle \forall\, i \in NG :: \left\langle \forall\, r \in R_i :: (r^{LST} \geq MAR_i) \Longrightarrow \Omega(r, \max(r^{EST}, MAR_i)) \in R'_i \right\rangle \right\rangle \\
RF3 &: R_1 = R'_1 \\
RF4 &: \langle \forall\, i : i \in NG : \langle \forall\, r' \in R'_i : REET_i^E \leq r'^{EST} + r'^{D} \rangle \rangle
\end{aligned}$$

Where the supporting function MAR, named for Maximum Ancestor REET, is defined as follows:

$$\begin{aligned}
&MAR_1 = -\infty \\
&MAR_i : NG \longrightarrow \mathbf{T} \triangleq \langle \max j : j \in ANCESTORS_i : REET_j^E \rangle
\end{aligned} \tag{5.5}$$

which returns the maximum end time of all the resources in plan REET that must complete before the resource for node i (ancestors of i).

The RF1 property states what resources **can** be in R'. Namely, if we take any resource $r' \in R'$ there must exist a resource $r \in R$ (the input resource set) that is like r' in every way except whose earliest start time (EST) attribute has been adjusted (with the Ω function). The RF2 property establishes what **must** be in R'. Specifically, it sets the condition that any resource $r \in R$ must satisfy to make it into R', namely its latest start time, r^{LST}, has to be larger than a certain value that has been computed to ensure feasibility. The expression RF3 states that all the resources for G_1 from the original set R are also in R'. RF4 describes the property of the $REET$ resources in R'. Note that r'^{EST} is equal to the second parameter of the Ω function.

We now show that the $REET\_FILTER$'s postcondition implies the following important properties:

Lemma 1 $\langle \forall i : i \in NG : REET_i \neq \perp \rangle \Longrightarrow$ *the REET plan is feasible.*

Proof. Applying the definition of feasibility (equation 5.4) to $REET$ we need to show that $\langle \forall\ i,j : i,j \in NG :\ i \rightsquigarrow j \Longrightarrow REET_i^E \leq REET_j^S \rangle$:

For any $i,j \in NG : i \rightsquigarrow j$ we know that:

$REET_j \in R_j'$ from RF0

$\Longrightarrow REET_j^S \geq MAR_i$ from RF1

$\Longrightarrow REET_j^S \geq REET_i^E$ from 5.5 ■

Lemma 2 *P is a feasible plan for $R \Longleftrightarrow P$ is a feasible plan for R'*

Proof. "$\Longrightarrow$"

Let P be a feasible plan for R:

base case:

$\left\langle \exists t : P_1^{EST} \leq t \leq P_1^S : \Omega(P_1,t) \in R_1' \right\rangle$ {e.g., $t = P_1^{EST}$}

induction hypothesis:

Suppose $\left\langle \forall i : 1 \leq i \leq k : \left\langle \exists t : P_i^{EST} \leq t \leq P_i^S : \Omega(P_i,t) \in R_i' \right\rangle \right\rangle$

1. $P_{k+1}^{LST} \geq P_{k+1}^S$ (basic property of a resource)

2. $\langle \forall j : 1 \leq j \leq k : P_{k+1}^S \geq P_j^E \rangle$ (from feasibility equation 5.4)

   $\Longrightarrow \langle \forall j : 1 \leq j \leq k : P_j^E \geq REET_j^E \rangle$ (induction hypothesis and RF4)

   $\Longrightarrow \langle \max j : 1 \leq j \leq k : P_j^E \rangle \geq MAR_{k+1}$

3. $\langle \max j : 1 \leq j \leq k : P_j^E \rangle = P_{1..k}^E$ (definition 4)

   $\Longrightarrow P_{k+1}^{LST} \geq P_{k+1}^S \geq P_{1..k}^E \geq MAR_{i+1}$ (2 & 3)

$\Longrightarrow \Omega(P_{k+1}, \max(P_{k+1}^{EST}, MAR_{k+1})) \in R'_{k+1}$ (RF2)

4. $P_{k+1}^{S} \geq P_{k+1}^{EST}$

   $\Longrightarrow \left\langle \forall i : 1 \leq i \leq k+1 : \left\langle \exists t : P_i^{EST} \leq t \leq P_i^{S} : \Omega(P_i, t) \in R'_i \right\rangle \right\rangle$ (by induction on k)

Thus, a derived version of P_{k+1} (via Ω) that is allowed to start at time P_{k+1}^{S} is in R'_{k+1} Therefore, P is a feasible plan for R'.

"$\Longleftarrow$"

follows immediately from RF1. ■

Lemma 3 *P is a feasible plan for $R' \Longrightarrow \langle \forall i : i \in NG : REET_i \neq \perp \rangle$*

Proof. P is feasible

$\Longrightarrow \forall i : i \in NG : P_i \neq \perp$ (trivial from feasibility equation 5.4)

$\Longrightarrow \forall i : i \in NG : P_i \in R'_i$ (definition of plan equation 5.2)

$\Longrightarrow \forall i : i \in NG : R'_i \neq \{\perp\}$

$\Longrightarrow \forall i : i \in NG : REET_i \neq \perp$ (RF4) ■

Theorem 1 *If there exists a feasible plan for the resource set R, then the $REET\_FILTER$ function of R will produce a feasible plan $REET$.*

Proof. Suppose there exists a feasible plan P for the resource set R. Then:

1. P is a feasible plan for $R \Longrightarrow P$ is a feasible plan for R' (lemma 2)

2. P is a feasible plan for $R' \Longrightarrow \langle \forall i : i \in NG : REET_i \neq \perp \rangle$ (lemma 3)

3. $\langle \forall i : i \in NG : REET_i \neq \perp \rangle \Longrightarrow$ the $REET$ plan is feasible (lemma 1)

   $\Longrightarrow$ the $REET$ plan is feasible (and the $REET\_FILTER$ produces $REET$).

■

Defining and proving the properties RF0-RF4 has been the hard part of the problem. Designing an algorithm that implements the $REET\_FILTER$ function based on its properties is rather straight forward. The following is a pseudo-code for one such implementation.

Algorithm 1 *(REET_FILTER)*

1. $R' = \emptyset$
2. **for** $i = 1$ **to** Q **do**:
3. $\quad REET_i := \perp$
4. **for** $r \in R_1$ **do**: // compute $REET_1$
5. $\quad R' := R' \cup \{r\}$ // $R_1 = R'_1$
6. $\quad$ **if** $REET_1^E > r^{EST} + r^D$ **then** $REET_1 := \{r, r^{EST} + r^D\}$
7. **for** $i = 2$ **to** Q **do**:
8. $\quad$ **for** $r \in R_i$ **do**: // at most M
9. $\quad\quad$ starttime $:= -\infty$
10. $\quad\quad$ **for** $k \in ANCESTORS_i$ **do** // at most W
11. $\quad\quad\quad$ **if** $REET_k \neq \perp \wedge r^{LST} \geq REET_k^E$ **then**
12. $\quad\quad\quad\quad$ starttime := max(starttime, max($REET_k^E, r^{EST}$))
13. $\quad\quad\quad$ **else**
14. $\quad\quad\quad\quad$ **break**; **continue** to next r // this r is disqualified
15. $\quad\quad$ r^{EST} := starttime
16. $\quad\quad$ $R' := R' \cup \{r\}$ // add r to set R'
17. $\quad\quad$ **if** $REET_i^E > r^{EST} + r^D$ **then**:
18. $\quad\quad\quad$ $REET_i := \{r, r^{EST} + r^D\}$

The algorithm iterates over all resources enforcing the RF0-RF4 conditions at each step. The run time of this algorithm is $O(Q * M * W)$ in the worst case, where $Q * M$ is the total number of resources and W is the maximum width of graph G (the inner-most loop). The space complexity is $Q * M$ in the worst case, which is $R' = R$.

5.2.2 Optimality

The second problem of interest is finding an optimal plan for an arbitrary directed acyclic computation graph. In general, finding an optimal plan requires constructing all the feasible plans for the computation graph G and the input resource set R and comparing their objective function values. In the worst case, if all the resources in R are schedulable, this will yield M^N possible feasible plans. We have reasons to believe that this problem is NP-complete, but have not formally proved it yet. The problem can be stated as follows:

Problem 2 *Find an optimal feasible plan for a set of resources R given a directed acyclic computation graph G and a maximum time horizon H.*

5.2.2.1 The Solution: Informal Description

The problem as stated is hard because it is very general. However, we have found that we can restrict the problem in two ways without significantly sacrificing its applicability. The new problem and the solution that we propose will apply to the class of resource configurations and on the time scale that are most likely to be of interest in the foreseeable future.

- The first difficulty with the original problem is the real nature of the time line on which it is specified. Because our resources can start on time ranges instead of fixed times, it becomes problematic when a range has infinite number of points any of which could be a solution. In the previous section we saw that this was not a problem for finding a feasible plan, because we needed only one. The algorithm for feasibility was always selecting a resource which was "a minimum" of a partial order of the end times of the resources that satisfied a particular stage of the construction. While it is possible to adapt the same rule as we used in feasibility, and assume that for a given resource it is always better to finish earlier to allow more possibilities for the subsequent nodes in the graph, we can still engineer a resource set that would require all possible resource configurations to be checked. Furthermore, a resource set could be chosen (this is the worst case) where all the resources could participate in some feasible plan and thus give us the M^N plans to compare.

  To reduce the number of possible feasible plans and make the problem tractable we **discretize** the time line T from 0 (now) to H (the time horizon of the whole computation) with a time step $\triangle t$, such that each step represents a segment of the real time line. The new time line is the set **DT** that stands for "discrete time line." We also transform the input resource set R into a new set DR by adjusting the resources' durations and start and end times to ensure that they fall on the discrete steps. This effectively groups the resources starting at the approximately the same time into the same discrete time slot which would enable us to compare them without inserting into a plan. For example, if two airline flights in reality could take off within 5 minutes of each other, and the discretization step the user chose is $\triangle t = 10$ minutes, then both resources will be adjusted to start on the same 10 minute mark.

- The second restriction that we place on the problem is the choice of computation graphs that a consumer could specify. We define a class of graphs we call *regular*, that have a nice recursive pattern that is most characteristic of the resource allocation problems that we believe to be likely to arise. Because of their structure such graphs yield to a clean algorithmic solution. On the other hand, further research into other types of graphs may be of interest in the future.

Figures 5.2 and 5.3 show two computation graphs. Regular graphs can be represented using a simple notation, where $\langle G_1, G_2 \rangle$ denotes a graph that can be divided into two subgraphs G_1 and G_2

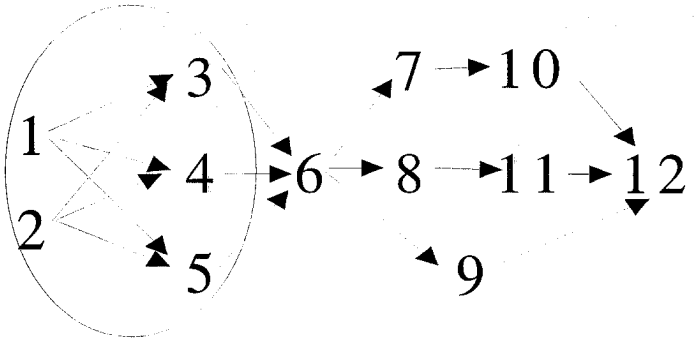


Figure 5.2: A regular graph

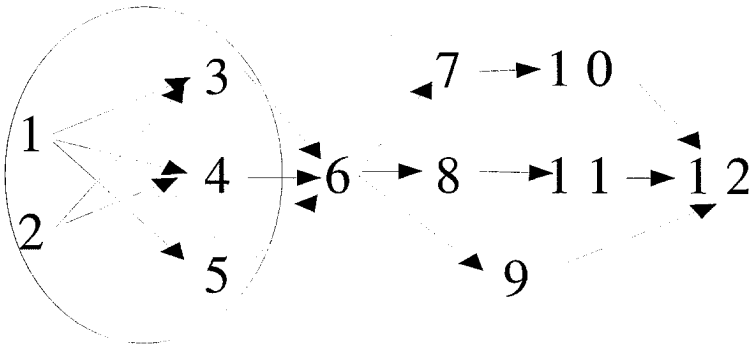


Figure 5.3: Not a regular graph.

that are in a temporal sequence with respect to each other; and $[G_1, .., G_k]$ represents a graph where $G_1, .., G_k$ are subgraphs with the same parents and same children. The figures 5.4 and 5.5 show a simple concurrent set $[i, .., j]$ and a sequential set $\langle i, i+1 \rangle$ respectively.

The computation graph G in Figure 5.1 could be given in this notation as

$$\langle \langle 1, [\langle 2, 6 \rangle, \langle [3, 4], 5 \rangle] \rangle, 7 \rangle$$

and the regular graph in Figure 5.2 can be represented as

$$\langle \langle \langle \langle [1, 2], [3, 4, 5] \rangle, 6 \rangle, [\langle 7, 10 \rangle, \langle 8, 11 \rangle, 9] \rangle, 12 \rangle$$

Note that the nodes $3, 4$ and 5 in Figure 5.2 are meant to be reserved concurrently and have the same parents, the nodes 1 and 2, and children, the node 6. Furthermore, the expressions $[1, 2]$ and $[3, 4, 5]$ have a temporal sequence, that is $[1, 2]$ must complete before $[3, 4, 5]$ could begin. Therefore we introduced the bracket notation that allows us to group them together and reason about them

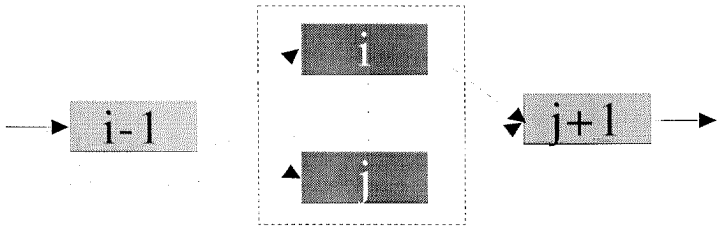


Figure 5.4: Concurrent set

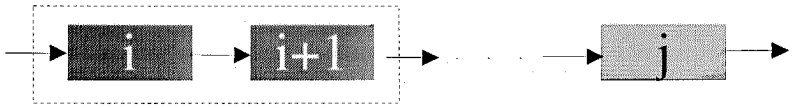


Figure 5.5: Sequential set

as single nodes. A non-regular graph cannot be represented in this notation for reasons that will become clear after a formal definition.

What we gain with discretization is tractability and we will show an algorithm that finds an optimal solution for directed acyclic graphs we called *regular*, that executes in $O(|R| * |G| * H^2)$ time and requires $O(|G| * H^2)$ space. What we give up is the certainty that the solution is optimal. That is, because of the discretization the algorithm might miss the best feasible plan. Thus, the optimality for which we will settle here is that of the best solution anyone could find in a reasonable time rather than the best possible.

We now state the optimal planning problem that we solve in this thesis.

Problem 3 *(Restatement) Find an optimal feasible plan for a set of resources DR given a regular directed acyclic computation graph G and a maximum time horizon H.*

The key to solving Problem 3 is in finding a method for selecting the resources from DR that would result in an optimal plan. One way to approach this is by first constructing partial plans and then merging them into single complete reservation plan. Since many partial plans could be constructed, we would like find and keep only those that have a chance of being part of an optimal plan, while forgetting the others. If the optimal plan could be constructed out of optimal partial plans then building such partial plans would enable us to compose the complete optimal plan. This property is called the *optimal substructure* property and we show that our problem does indeed exhibit it.

We have defined regular graphs as those that can be represented using the bracket notation above. The reason why we chose such graphs is 1) because we have found a way to find optimal solutions to subgraphs denoted with the brackets and 2) because a complete reservation plan for such graphs can be constructed out partial plans for the subgraphs that have overlapping structure. By finding optimal partial plans to these subgraphs in a particular order we can design a dynamic programming algorithm that would solve the optimization problem in polynomial time.

In such an algorithm, we repeatedly seek the optimal partial plan for a certain subgraph of the complete computation graph. The newly discretized time line allows us to compute optimal partial plans for all possible times. If the computation is carried out from "the bottom up," namely the optimal partial plans for small subgraphs are found first, then their values could be used directly in the optimal plans for bigger and bigger subgraphs until the complete optimal reservation plan is computed.

5.2.2.2 The Solution Strategy: Formal Description

This transformation from R toDR can be described by the following function:

Definition 8 *(Discretization)*

1. $\mathbf{DT} \triangleq \mathbf{N} \cup \{-\infty, +\infty\}$ *our time scale where 0 is "now"*
2. $DISCRETIZE : R \longrightarrow DR$
3. $TIMERANGE_r : R \longrightarrow 2^{DT} \triangleq \left\{t : t \in \mathbf{DT} \wedge r^{EST} \leq t \leq r^{LST}\right\}$
4. $\Theta : R \times \mathbf{DT} \times \mathbf{DT} \times \mathbf{DT} \longrightarrow DR \triangleq \{r, est, lst, d\} \longrightarrow dr : r = dr$ *except* $dr^{EST} = est$, $dr^{LST} = lst$, $dr^D = d$
5. $D\Pi_{NS}$ *is the set of all feasible partial plans over set NS on DR*

We define the postcondition of $DISCRETIZE$ to be as follows:

$$\left\langle \forall r \in R :: \left\langle \begin{array}{c} \exists dr \in DR : dr = \Theta(r, \min(TIMERANGE_r), \\ \max(TIMERANGE_r), \min\{t \in \mathbf{DT} : t \geq r^D\}) \end{array} \right\rangle \right\rangle \wedge |R| = |DR| \tag{5.6}$$

Definition 9

$$\begin{array}{lcl} GRAPH & := & \langle PGRAPH, PGRAPH \rangle \\ PGRAPH & := & i \in NG \\ & & \mid \langle PGRAPH, PGRAPH \rangle \\ & & \mid [PGRAPH, PGRAPH \ \{, PGRAPH\}^*] \end{array}$$

Definition 10 *Let* $t_o, t_1 \in \mathbf{DT}$ *and GR be a GRAPH representation of a computation graph G.*

1. $NODES(GR) : GRAPH \longrightarrow 2^{NG}$ *is the set of nodes from NG that are represented in the GRAPH structure GR.*

2. $OptValue[GR, t_o, t_1] \triangleq \langle \max p : p \in D\Pi_{NODES(GR)} \wedge p^S \geq t_0 \wedge p^E \leq t_1 : p^{VAL} \rangle$ *is the value of a partial plan over* $NODES(GR)$ *with the highest objective function value that starts and ends between the times* t_o *and* t_1.
3. $OptPlan[GR, t_o, t_1]$ *is the actual partial plan that corresponds to the* $OptValue[GR, t_o, t_1]$
4. $OptValue[GR, 0, H]$ *is the value of the optimal complete reservation plan.*
5. $OptPlan[GR, 0, H]$ *is the actual complete reservation plan*

Definition 11 *A graph G is said to be* ***regular*** *if and only if for its GRAPH representation GR*

case 1: $GR = \langle GR_1, GR_2 \rangle$ *implies*

$$NODES(GR_1) \subset NODES(GR)$$
$$\wedge \quad NODES(GR_2) \subset NODES(GR)$$
$$\wedge \quad NODES(GR_1) \cup NODES(GR_2) = NODES(GR)$$
$$\wedge \quad NODES(GR_1) \cap NODES(GR_2) = \emptyset$$
$$\wedge \quad (GR_1 \text{ and } GR_2 \text{ are regular})$$
$$\wedge \quad (GR_3 = Leaves(GR_1) \cup Roots(GR_2) \text{ is isomorphic to a CBG with the edges from leaves to roots.})$$

where CBG stands for a Complete Bipartite Graph.

case 2: $GR = [GR_1, .., GR_k]$ *implies*

$$\forall i : 1 \leq i \leq k : GR_i \text{ is regular}$$

case 3: $GR = \{k\}$ *we define to be regular.*

Lemma 4 *Let* P_{NS} *be a feasible partial plan where* $NS \subseteq NG \wedge |NS| > 1$ *is regular with its GRAPH representation NSR . Then:*

1. $NSR = \langle NSR_1, NSR_2 \rangle \implies P_{NODES(NSR_1)}$ *and* $P_{NODES(NSR_2)}$ *are feasible partial plans such that* $P^E_{NODES(NS_1)} \leq P^S_{NODES(NS_2)}$
2. $NSR = [NSR_1, .., NSR_k] \implies \langle \forall i : 1 \leq i \leq k : P_{NODES(NS_i)} \text{ is feasible} \rangle$

Proof.

1. $\langle \forall r_1 : r_1 \in P_{NODES(NSR_1)} :: \langle \forall r_2 \in P_{NODES(NSR_2)} : r_1^E \geq r_2^S \rangle \rangle$ (from definition 11 case 1 and the feasibility equation 5.4)

   $\implies \langle \max r : r \in P_{NODES(NSR_1)} : r^E \rangle \leq \langle \min r : r \in P_{NODES(NSR_2)} : r^S \rangle$

   $\implies P^E_{NODES(NSR_1)} \leq P^S_{NODES(NSR_2)}$ (from definition of P^E and P^S)

2. immediate from the feasibility equation 5.4.

■

Theorem 2 *An optimal feasible plan for a regular directed acyclic graph over DR has an optimal substructure.*

Proof. Suppose that a complete optimal reservation plan $OptPlan[G, 0, H]$ exists. Its objective function value is $OptValue[G, 0, H]$. Let $OptPlanGR$ be the $GRAPH$ representation of G (because G is regular). Then, $OptPlanGR = \langle NSR_1, NSR_2 \rangle$ (because G is connected). Let $NS_1 = NODES(NSR_1)$ and $NS_2 = NODES(NSR_2)$ and P_{NS_1} and P_{NS_2} be the corresponding partial plans.

Let $t \in DT$ such that $P^E_{NS_1} \leq t \leq P^S_{NS_2}$ (the existence t is guaranteed by lemma 4)

We want to show that $(P_{NS_1} = OptPlan[NSR_1, 0, t]) \wedge (P_{NS_2} = OptPlan[NSR_2, t, H])$.

Suppose $P_{NS_1} \neq OptPlan[NSR_1, 0, t]$

$\Longrightarrow \left(\exists p \in D\Pi_{NS_1} : p^S \geq 0 \wedge p^E \leq t : p^{VAL} > P^{VAL}_{NS_1}\right)$

$\Longrightarrow \dot{P} = p \cup P_{NS_2}$ is a complete feasible plan (from feasibility equation 5.4)

$\Longrightarrow \dot{P}^{VAL} = p^{VAL} + P^{VAL}_{NS_2}$ (from definition 5.3)

$\Longrightarrow \dot{P}^{VAL} > OptValue[G, 0, H]$

$\Longrightarrow$ contradiction.

The argument is similar for the other two cases: $P_{NS_2} \neq OptPlan[NSR_2, t, H]$ and both P_{NS_1} and P_{NS_2} are suboptimal. Therefore, $(P_{NS_1} = OptPlan[NSR_1, 0, t]) \wedge (P_{S_2} = OptPlan[NSR_2, t, H])$. Furthermore, this argument can be recursively applied to the partial plans for NSR_1 and NSR_2 because the graph is regular. ■

The Theorem 2 shows that for regular graphs an optimal plan must be constructed out of the optimal partial plans. What remains to be shown is how to compute the optimal partial plans for each of the $GRAPH$ structures of a $GRAPH$ representation of a complete reservation plan. Namely, we need to compute $OptValue[NSR, t_o, t_1]$, where NSR can be any of the three cases in $GRAPH$'s BNF.

Definition 12 *Let NS be a concurrent* $[i, ..j]$*, a sequence* $< i, j >$ *or a node* i*. Let* $0 \leq t_o, t_1 \leq H$*. Then*

$$\begin{aligned}
OptValue[i, t_o, t_1] &\triangleq \left\langle \max r, t : r \in R^t_i \wedge t_o \leq t \leq t_1 \wedge t - r^D \geq t_o : r^{VAL} \right\rangle \\
OptValue[[i, ..j], t_o, t_1] &\triangleq \langle \textstyle\sum k : i \leq k \leq j : OptValue[k, t_o, t_1] \rangle \\
OptValue[\langle i, j \rangle, t_o, t_1] &\triangleq \langle \max t : t_0 < t < t_1 : OptValue[i, t_o, t] + OptValue[j, t, t_1] \rangle
\end{aligned} \tag{5.7}$$

Recall that $R^t \subseteq R$ is the set of all resources capable of ending at time t. Also, the function $OptValue[NSR, t_o, t_1]$ represents the highest value partial plan for NSR that starts and ends between t_o and t_1.

The $OptValue$ is a recursive function, whose value for any particular partial plan depends on the values of the smaller partial plans. The base case is the value of a partial plan for a single node

This graph is regular and therefore our reservation model can be applied. The objective function for this type of reservation may involve the quality of the facilities, professionals and a budget for the whole operation.

Chapter 7

Conclusion

7.1 Contributions

I have formulated a new distributed resource allocation problem that I expect to be of significant interest in the near future. I have proposed new algorithms to find feasible and optimal solutions over the space of problem instances that have immediate applications, such as crisis management and event planning.

7.2 Future Work

The space of problems that I have explored in this thesis was designed to solve specific problems in resource allocation. Thus, the model I have created is, while adequate, is not rich enough to handle some more general problems. For example, the choice of a directed acyclic graph to specify the temporal properties of a computation is insufficient to specify more precise constraints such as "resource A must start within 10 minutes of resource B"

To be able to solve the most general version of the resource allocation problem a model that allows arbitrary predicates between the resource types is needed. This is the topic of my doctorate research.

Bibliography

[1] *The Infospheres Infrastructure version 2.0 User's Guide.*

[2] *An Introduction to the Mathematics of Financial Derivatives.* Academic Press, 1996.

[3] R.G. Herrtwich H. Wittig C. Vogt, L.C.Wolf. Heirat - quality-of-service management for distributed multimedia systems. *Multimedia Systems*, 1998.

[4] Microsoft Corp. Dcom architecture. Technical report, Microsoft Corp., 1999.

[5] C. Fetzer and F. Cristian. An optimal internal clock synchronization algorithm. In *Proceedings of the 10th Annual IEEE Conference on Computer Assurance*, pages 187–196, Gaithersburg, MD, June 1995. http://www-cse.ucsd.edu/users/cfetzer/OCS/ocs.html.

[6] C. Kesselman I. Foster. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 1997.

[7] Sun Microsystems Inc. Jini connection technology overview. Technical report, Sun Microsystems Inc., 1999.

[8] Object management Group(OMG). *The Common Object Request Broker: Architecture and Specification (CORBA), revision 2.0.* Object Management Group (OMG), 1998.